General Features of Interrupts

- All interrupting systems must have the:
  1. Ability for hardware to request action from computer.
  2. Ability for computer to determine the source.
  3. Ability for computer to acknowledge the interrupt.

- To arm (disarm) a device means to enable (shut off) the source of interrupts.

- To enable (disable) means to allow (postpone) interrupts at this time.

Sequence of Events During Interrupt

1. Hardware needs service (busy-to-done) transition.
2. Flag is set in one of the I/O status registers.
   (a) Interrupting event sets the flag (ex., STAF=1).
   (b) The device is armed (ex., STAI=1).
   (c) Microcomputer interrupts are enabled (ex., I=0).
3. Thread switch.
   (a) Microcomputer finishes current instruction.
   (b) All registers are pushed onto the stack.
   (c) Vector address is obtained and put into the PC.
   (d) Microcomputer sets I=1.
4. Execution of the ISR.
5. Return control back to the thread that was running.

6811 Stack Before and After an Interrupt

Before the interrupt:
- PC
- SP
- RTI
- Old X
- Old Y
- Old PC

After the interrupt:
- PC
- SP
- RTI
- Old X
- Old Y
- Old PC
### 6811 Interrupts

- **6811** has two external requests, TRQ and XTRQ.
- Other interrupt sources include:
  - A STRA interrupt
  - Three input capture interrupts
  - Five output compare interrupts
  - Three timer interrupts (timer overflow, RTI, pulse accumulator)
  - Two serial port interrupts (SCI and SPI)

- Interrupts have a fixed priority, but can elevate one to highest priority using hardware priority interrupt (HPRIO) register.
- XIRQ is highest-priority device and has separate vector and enable bit (X).
- Once X bit is cleared, software cannot disable it.
- XIRQ handler sets X and I, and restores with rti.

### 6811 Interrupt Vectors and Priority

<table>
<thead>
<tr>
<th>Vector</th>
<th>Interrupt Source</th>
<th>Enable</th>
<th>Arm</th>
</tr>
</thead>
<tbody>
<tr>
<td>$FFFF</td>
<td>Power on reset</td>
<td>Always</td>
<td>Always highest</td>
</tr>
<tr>
<td>$FFFE</td>
<td>Hardware reset</td>
<td>Always</td>
<td>Always</td>
</tr>
<tr>
<td>$FFFC</td>
<td>COP clk monitor fail</td>
<td>Always</td>
<td>OPTION.CME=1</td>
</tr>
<tr>
<td>$FFFA</td>
<td>COP failure</td>
<td>Always</td>
<td>CONFIG.NOCOP=0</td>
</tr>
<tr>
<td>$FFFF</td>
<td>Nonmaskable XIRQ</td>
<td>X=0</td>
<td>External hardware</td>
</tr>
<tr>
<td>$FFFF</td>
<td>External IRQ</td>
<td>I=0</td>
<td>External hardware</td>
</tr>
<tr>
<td>$FFFF</td>
<td>Parallel I/O, STAF</td>
<td>I=0</td>
<td>PIO.STAI=1</td>
</tr>
<tr>
<td>$FFFF</td>
<td>Real time int., RTIF</td>
<td>I=0</td>
<td>TMSK2.RTI=1</td>
</tr>
<tr>
<td>$FFFF</td>
<td>Imp capture 1, IC1F</td>
<td>I=0</td>
<td>TMSK1.IC1I=1</td>
</tr>
<tr>
<td>$FFFF</td>
<td>Imp capture 2, IC2F</td>
<td>I=0</td>
<td>TMSK1.IC2I=1</td>
</tr>
<tr>
<td>$FFFF</td>
<td>Imp capture 3, IC3F</td>
<td>I=0</td>
<td>TMSK1.IC3I=1</td>
</tr>
<tr>
<td>$FFFF</td>
<td>Outp compare 1, OC1F</td>
<td>I=0</td>
<td>TMSK1.OC1I=1</td>
</tr>
<tr>
<td>$FFFF</td>
<td>Outp compare 2, OC2F</td>
<td>I=0</td>
<td>TMSK1.OC2I=1</td>
</tr>
<tr>
<td>$FFFF</td>
<td>Outp compare 3, OC3F</td>
<td>I=0</td>
<td>TMSK1.OC3I=1</td>
</tr>
<tr>
<td>$FFFE</td>
<td>Outp compare 4, OC4F</td>
<td>I=0</td>
<td>TMSK1.OC4I=1</td>
</tr>
<tr>
<td>$FFED</td>
<td>Outp compare 5, OC5F</td>
<td>I=0</td>
<td>TMSK1.OC5I=1</td>
</tr>
<tr>
<td>$FFDC</td>
<td>Timer overflow, TOF</td>
<td>I=0</td>
<td>TMSK2.TO1=1</td>
</tr>
<tr>
<td>$FFDD</td>
<td>Pulse accum overflow</td>
<td>I=0</td>
<td>TMSK2.PAOV1=1</td>
</tr>
<tr>
<td>$FFDA</td>
<td>Pulse accum inp edge</td>
<td>I=0</td>
<td>TMSK2.PAI1=1</td>
</tr>
<tr>
<td>$FFD8</td>
<td>SPI complete, SPIF</td>
<td>I=0</td>
<td>SPCR.SPIE=1</td>
</tr>
<tr>
<td>$FFD6</td>
<td>Rx data reg full, RDRF</td>
<td>I=0</td>
<td>SCCR2.RIE=1</td>
</tr>
<tr>
<td>$FFD6</td>
<td>Rx overrun, OVRN</td>
<td>I=0</td>
<td>SCCR2.RIE=1</td>
</tr>
<tr>
<td>$FFD6</td>
<td>Tx data reg empty, TDRE</td>
<td>I=0</td>
<td>SCCR2.TIE=1</td>
</tr>
<tr>
<td>$FFD6</td>
<td>Tx complete, TC</td>
<td>I=0</td>
<td>SCCR2.TCIE=1</td>
</tr>
<tr>
<td>$FFD6</td>
<td>Idle line detect, IDLE</td>
<td>I=0</td>
<td>SCCR2.ILIE=1</td>
</tr>
<tr>
<td>$FFFF</td>
<td>Illegal opcode trap</td>
<td>Always</td>
<td>Always</td>
</tr>
<tr>
<td>$FFFF</td>
<td>Software interrupt SWI</td>
<td>Always</td>
<td>Always lowest</td>
</tr>
</tbody>
</table>
### Slide 9

**Setting Interrupt Vectors**

- **org $FFF0**
- **fdb RTIHAN**  
  Pointer to real time interrupt handler
- **org $FFF2**
- **fdb IRQHAN**  
  Pointer to external IRQ and STRA handler
- **org $FFF4**
- **fdb XIRQHAN**  
  Pointer to external XIRQ handler
- **fdb RESETHAN**  
  Pointer to reset handler

### Slide 10

**6811 Pseudo-Vectors**

<table>
<thead>
<tr>
<th>6811 Vector</th>
<th>Vector Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>$FFD6</td>
<td>SCI</td>
<td>$00C4-$00C6</td>
</tr>
<tr>
<td>$FFD8</td>
<td>SPI</td>
<td>$00C7-$00C9</td>
</tr>
<tr>
<td>$FFDA</td>
<td>Pulse accum inp edge</td>
<td>$00CA-$00CC</td>
</tr>
<tr>
<td>$FFDC</td>
<td>Pulse accum overflow</td>
<td>$00CD-$00CF</td>
</tr>
<tr>
<td>$FFDE</td>
<td>Timer overflow, TOF</td>
<td>$00D0-$00D2</td>
</tr>
<tr>
<td>$FFE0</td>
<td>Outp compare 5, OC5F</td>
<td>$00D3-$00D5</td>
</tr>
<tr>
<td>$FFE2</td>
<td>Outp compare 4, OC4F</td>
<td>$00D6-$00D8</td>
</tr>
<tr>
<td>$FFE4</td>
<td>Outp compare 3, OC3F</td>
<td>$00D9-$00DB</td>
</tr>
<tr>
<td>$FFE6</td>
<td>Outp compare 2, OC2F</td>
<td>$00DC-$00DE</td>
</tr>
<tr>
<td>$FFE8</td>
<td>Outp compare 1, OC1F</td>
<td>$00DF-$00E1</td>
</tr>
</tbody>
</table>

### Slide 11

**6811 Pseudo-Vectors (cont)**

<table>
<thead>
<tr>
<th>6811 Vector</th>
<th>Vector Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>$FFEA</td>
<td>Inp capture 3, IC3F</td>
<td>$00E2-$00E4</td>
</tr>
<tr>
<td>$FFEC</td>
<td>Inp capture 2, IC2F</td>
<td>$00E5-$00E7</td>
</tr>
<tr>
<td>$FFEE</td>
<td>Inp capture 1, IC1F</td>
<td>$00E8-$00EA</td>
</tr>
<tr>
<td>$FFF0</td>
<td>Real time int., RTIF</td>
<td>$00EB-$00ED</td>
</tr>
<tr>
<td>$FFF2</td>
<td>External IRQ, STAF</td>
<td>$00EE-$00F0</td>
</tr>
<tr>
<td>$FFF4</td>
<td>Nonmaskable XIRQ</td>
<td>$00F1-$00F3</td>
</tr>
<tr>
<td>$FFF6</td>
<td>Software interrupt SWI</td>
<td>$00F4-$00F6</td>
</tr>
<tr>
<td>$FFF8</td>
<td>Illegal opcode trap</td>
<td>$00F7-$00F9</td>
</tr>
<tr>
<td>$FFFA</td>
<td>COP failure</td>
<td>$00FA-$00FC</td>
</tr>
<tr>
<td>$FFFC</td>
<td>COP clk monitor fail</td>
<td>$00FD-$00FF</td>
</tr>
</tbody>
</table>

### Slide 12

**Setting Interrupt Pseudo-Vectors**

- **ldaa #$7E**  
  Opcode for JMP
- **staa $00EB**
- **ldx #RTIHAN**  
  Pointer to real time interrupt handler
- **stx $00EC**  
  JMP RTIHAN
- **ldaa #$7E**  
  Opcode for JMP
- **staa $00EE**
- **ldx #IRQHAN**  
  Pointer to external IRQ and STRA handler
- **stx $00EF**  
  JMP IRQHAN
- **ldaa #$7E**  
  Opcode for JMP
- **staa $00F1**
- **ldx #XIRQHAN**  
  Pointer to external IRQ and STRA handler
- **stx $00F2**  
  JMP XIRQHAN
External Interrupt Design Approach

- First, identify status signal that indicates the busy-to-done state transition.
- Next, connect the I/O status signal to a microcomputer input that can generate interrupts.

<table>
<thead>
<tr>
<th>MC68HC11A8</th>
<th>I/O Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRA</td>
<td>Status</td>
</tr>
<tr>
<td>PA2-0</td>
<td></td>
</tr>
</tbody>
</table>

Polled Versus Vectored Interrupts

- Vectored interrupts - each interrupt source has a unique interrupt vector address.
- Polled interrupts - multiple interrupt sources share the same interrupt vector address.
  - Minimal polling - check flag bit that caused interrupt.
  - Polling for 0s and 1s - verify entire status register.

Interrupting Software

1. Ritual - executed once, disable interrupts during, initialize globals, set port dir, set port interrupt ctrl reg, clear interrupt flag, arm device, and enable interrupts.
2. Main program - initialize SP, execute ritual, interacts with ISRs via global data (ex. FIFO queue).
3. ISR(s) - determine interrupt source, implement priority, acknowledge (clear the flag) or disarm, exchange info w/main program via globals, execute rti to exit.
4. Interrupt vectors - in general purpose processors vectors in RAM, in embedded systems usually in ROM.

External I/O Device(s) Connected to the Microcomputer

<table>
<thead>
<tr>
<th>MC68HC11A8</th>
<th>I/O Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRA</td>
<td>Status</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MC68HC11A8</th>
<th>I/O Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRQ</td>
<td>Status</td>
</tr>
<tr>
<td>6821</td>
<td></td>
</tr>
</tbody>
</table>
Example of a Vectored Interrupt

TimeHan ldaa #$80 ;TOF is bit 7
staa TFLG2 ;clear TOF
;r*Timer interrupt calculations*
rti

ExtHan ldaa PIDC
ldaa PORTCL ;clear STAF
;r*External interrupt calculations*
rti
org $FFDE ;timer overflow
fdb TimeHan
org $FFF2 ;IRQ external
fdb ExtHan

Example of a Polled Interrupt

ExtHan ldaa PIDC ;which one
bita #$80 ;STAF?
bne STAFHan
ldaa OtherStatus1
bita #$80 ;External?
bne OtherHan
swi ;error
STAFHan ldaa PORTCL ;clear STAF
;r*STAF interrupt calculations*
rti

OtherHan ldaa OtherData
;r*Other interrupt calculations*
rti
org $FFF2 ;IRQ external
fdb ExtHan

Keyboard Interface Using Interrupts

1. IRQ
2. STROBE

Interrupting Keyboard Software in Assembly

; PC6-PC0 inputs = keyboard DATA
; STRA=STROBE interrupt on rise
; 6 STA1 1 Interrupts armed
; 5 CWDM 0 Normal outputs
; 4 HNDs 0 No handshake
; 3 OIN 0
; 2 PLS 0 STRB not used
; 1 EGA 1 STAF set on rise of READY
; 0 INVB 0 STRB not used

Init sei ;Make this atomic
ldaa #$80 ;PC7 is an output
staa DDRC ;PC6-0 inputs
ldaa #$42
staa PIOC
Interrupting Keyboard Software in Assembly

```assembly
ldaa PIOC ; clears STAF
ldaa PORTCL
clr PORTC ; Make PC7=0
jsr InitFifo
cli ; Enable IRQ
rts
ExtHan ldaa PIOC ; poll STAF
bmi KeyHan
swi ; error
KeyHan ldaa PORTCL ; clear STAF
jsr PutFifo
rti
org $FFF2 ; IRQ external
fdb ExtHan
```

Printer Interface Using IRQ Interrupts

```
// PC6-PC0 inputs = key DATA, STRA=STROBE interrupt
void Init(void){
  unsigned char dummy;
  asm("sei");
  PIOC=0x42; // EGA=1, STAI
  DDRC=0x80; // STRA=STROBE
  PORTC=0x00; // PC7=0
  dummy=PIOC; dummy=PORTCL;
  InitFifo();
  asm(" cli");
  #pragma interrupt_handler ExtHan()
  void ExtHan(void){
    if((PIOC & STAF)==0)asm(" swi");
    PutFifo(PORTCL);} // ack
```

Printer Interface Helper Routines in Assembly

`;*****goes in RAM***************`
OK rmb 1 ; 0=busy, 1=done
Line rmb 20 ; ASCII, end with 0
Pt rmb 2 ; pointer to Line
`;*****goes in ROM***************`
; Input RegX=> string
Fill ldy #Line; RegX=> string
sty Pt ; initialize pointer
Floop ldaa 0,X ; copy data
  staa 0,Y
  inx
  iny

Printer Interface Using IRQ Interrupts

```
// SEGA=1, STAI
DDRC=0x80; // STRA=STROBE
```

Printer Interface Helper Routines in Assembly

`;*****goes in RAM***************`
OK rmb 1 ; 0=busy, 1=done
Line rmb 20 ; ASCII, end with 0
Pt rmb 2 ; pointer to Line
`;*****goes in ROM***************`
; Input RegX=> string
Fill ldy #Line; RegX=> string
sty Pt ; initialize pointer
Floop ldaa 0,X ; copy data
  staa 0,Y
  inx
  iny
Printer Interface Helper Routines in Assembly

```assembly
Slide 25

; tsta ; end?

bne Floop
clr OK
rts

; Return RegA=data

Get  ldx Pt

ldaa 0,X ; read data

inx

stx Pt

rts
```

Printer Interface Initialization Routines in Assembly

```assembly
Slide 27

; PC6-PC0 outputs = printer DATA
; STRA=READY interrupt on rise
; Input RegX=>string

Init  sei ; Make this atomic

bsr Fill ; Init global

ldaa #$FF ; PC7 is an output

staa DDRC ; PC6-0 outputs

ldaa #$5E

staa PIOC

bsr Get ; start first

staa PORTCL

cli ; Enable IRQ

rts
```

Printer Interface Helper Routines in C

```plaintext
Slide 26

// PC6-PC0 outputs = printer DATA
// STRA=READY interrupt on rise
// STRB=START pulse out

unsigned char OK; // 0=busy, 1=done
unsigned char Line[20]; // ASCII data
unsigned char *Pt; // pointer to line

void Fill(unsigned char *p){
    Pt=&Line[0];
    while((Pt++)==(*p++)); // copy
    Pt=&Line[0]; // initialize pointer
    OK=0;
}

unsigned char Get(void){
    return(*Pt++);
}
```

Printer Interface ISR in Assembly

```assembly
Slide 28

ExtHan ldaa PIOC ; poll STAF?

bmi PrtHan

swi ; error

PrtHan bsr Get

tsta

beq Disarm

staa PORTCL ; start next

bra Done

Disarm ldaa #$1E ; STAI=0

staa PIOC

inc OK ; line complete

Done rti

org $FFF2 ; IRQ external

fdb ExtHan
```
**Printer Interface in C**

```c
void Init(unsigned char *thePt){
    asm("sei"); // make atomic
    Fill(thePt); // copy data into global
    DDRC=0xFF; // Port C outputs
    PIOC=0x5E; // arm out handshake
    PORTCL=Get(); // start first
    asm(" cli");}
#else
    interrupt_handler
void ExtHan(void){
    if((PIOC & STAF)==0)asm(" swi");
    if(data=Get())
        PORTCL=data; // start next
    else{
        PIOC=0x1E; // disarm
        OK=1;}} // line complete
#endif
```

**Power System Interface in Assembly**

```
RITUAL ldaa #0 Backup power initially off
staa PORTB Set the flip flop, make XIRQ=1
lda #1
staa PORTB Ready to receive rising edge
lda #10 Enable XIRQ, Disable IRQ
tap
rts Back to main thread
* Software can only enable XIRQ, not disable it.
XIRQHAN ldaa #2
    staa PORTB Enable BackUp, ack XIRQ
lda #3
    staa PORTB
rti
    org $FFF4
    fdb XIRQHAN XIRQ interrupt vector
```

**Power System Interface Using XIRQ**

```
Power System Interface Using XIRQ

XIRQ 6811/6812 Up Low
PB0
PB1

Power System
Too Low

74HC74
Q D
S R

Enable Backup
```

**Power System Interface in C**

```c
/* Power System interface
XIRQ requested on a rise of TooLow
PB0, negative logic pulse, will acknowledge XIRQ
PB1=1 will activate backup power */
#endif

void PowerLow(void){
    PORTB=2; PORTB=3;
}

void Ritual(void){
    // Port B outputs
    PORTB=0; PORTB=1; // Make XIRQ=1
    asm(" ldaa #0\x10\n" " tap");
}
```